HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 10016628-1

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Donald C. Soltis, Jr., et al.

Confirmation No.: 3827

Application No.: 10/092,670

Examiner: Tonia L. Meonske

Filing Date: March 6, 2002

Group Art Unit: 2181

Title: Using Thread Urgency In Determining Switch Events In A Temporal Multithreaded Processor Unit

Mail Stop Appeal Brief-Patents
Commissioner For Patents
PO Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF APPEAL BRIEF

Transmitted herewith is the Appeal Brief in this application with respect to the Notice of Appeal filed on August 7, 2006.

The fee for filing this Appeal Brief is (37 CFR 1.17(c)) $500.00.

(complete (a) or (b) as applicable)

The proceedings herein are for a patent application and the provisions of 37 CFR 1.136(a) apply.

☒ (a) Applicant petitions for an extension of time under 37 CFR 1.136 (fees: 37 CFR 1.17(a)-(d)) for the total number of months checked below:

☒ 1st Month $120	☐ 2nd Month $450	☐ 3rd Month $1020	☐ 4th Month $1590

☐ The extension fee has already been filed in this application.

☐ (b) Applicant believes that no extension of time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition and fee for extension of time.

A Check for $120 for the extension of time is enclosed.
Please charge to Deposit Account 08-2025 the sum of $ 500. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

☒ I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to:
Commissioner for Patents, Alexandria, VA 22313-1450
Date of Deposit: November 7, 2006

OR

☐ I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number (571)273-8300.
Date of facsimile:
Typed Name: Keiko Morioka
Signature:

Respectfully submitted,
Donald C. Soltis, Jr., et al.
By
Curtis A. Vock
Attorney/Agent for Applicant(s)
Reg No.: 38,356
Date: November 7, 2006
Telephone: (720) 931-3011

11/13/2006 TBESHAH1 00000049 10092670
01 FC:1251 120.00 OP

Rev 10/05 (AplBrief)

TW AF

PATENT
Attorney Docket No.: 10016628-1
Sent Via First Class Mail

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s)	Donald C. Soltis Jr.	Confirmation No.	3827
Serial No.	10/092,670	Group Art No.	2181
Filed	March 6, 2002	Examiner	Meonske, Tonia L
For	Using Thread Urgency In Determining Switch Events In A Temporal Multithreaded Processor Unit		

November 7, 2006

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

Dear Sir:

In accord with 37 C.F.R. § 41.37, and fully responsive to the Office Action of May 5, 2006, Appellants hereby file their appeal brief in support of their Appeal in the above-identified matter (hereinafter the '670 Application). A notice of appeal, with appropriate fee of $500 as required by 37 C.F.R. §§41.31, 41.20(b)(1), was filed on August 7, 2006. Three copies of this brief are enclosed. The $500 fee for this appeal brief, as required by 37 C.F.R. §41.20(b)(2), is also filed herewith. This appeal brief is timely filed within two months (plus extensions) of the mailing of the notice of appeal.

11/15/2006 TBESHAH1 00000004 082025 10092670
01 FC:1402 500.00 DA

(1) Real party in interest.

The real party in interest for this appeal is Hewlett-Packard Development Company, L.P. (HPDC), a limited partnership established under the laws of the State of Texas and having a principal place of business at 20555 S.H. 249 Houston, TX 77070, U.S.A. HPDC is a Texas limited partnership and is a wholly-owned affiliate of Hewlett-Packard Company, a Delaware Corporation, headquartered in Palo Alto, CA. The general or managing partner of HPDC is HPQ Holdings, L.L.C. Evidence of this assignment, which was recorded on September 30, 2003, may be found at reel/frame 014061/0492.

(2) Related appeals and interferences.

No other appeals or interferences are currently known to Appellant that will directly affect, be directly affected by, or have a bearing on the decision to be rendered by the Board of Patent Appeals and Interferences in the present appeal.

(3) Status of claims.

Claims 1-19 are pending in the '670 Application. Applicants appeal all claims 1-19. Claims 1-19 now stand rejected under 35 U.S.C. §102(b) as being anticipated by Keckler, Stephen W., et al., titled "Concurrent Event Handling through Multithreading", 1999, IEEE Transactions on Computers, volume 48, NO. 9, pages 903-916 (hereinafter "Keckler").

(4) Status of amendments.

The '670 Application was filed on March 6, 2002. A first office action was mailed on August 24, 2004, to which a response was filed and entered November 22, 2004. On February 7, 2005, a final office action was mailed, to which a response was filed on March 31, 2005. An advisory action was mailed on April 19, 2005, to which a request for continued examination was filed on May 6, 2005. A non-final rejection was mailed on July 26, 2005, to which a response was filed on October 26, 2005. A miscellaneous action was filed on January 9, 2006, to which a response was filed on January 31, 2006. A final rejection was mailed on May 5, 2006, prompting this appeal. Claims 1-19 are currently pending, of which claims 3, 4, 6 and 8-12 are original (without claim amendment during

prosecution). Claims 7 and 19 were amended in the response of November 22, 2004. Claims 1, 2, 5 and 13-19 were amended in the response of March 31, 2005.

(5) Summary of claimed subject matter.

The inventions of claims 1-13 relate to a method for determining thread switch points within pipeline execution units of a processor. The flowchart 100 of FIG. 3 illustrates exemplary thread-controller operation for determining switch points within pipeline execution units of a processor. The method of claim 1 is also understood in connection with paragraphs [0006]-[0010] and [0023]-[0026] of the specification. Processing of a first thread is monitored within the pipeline execution units of the processor. A first urgency indicator of the first thread is based upon execution of the first thread within the pipeline execution units. At a possible switch point for the first thread, the first thread is deactivated, or not, based upon the first urgency indicator. See for example, paragraph [0024] of the specification.

Claims 14-19 relate to a processor for processing multi-threaded program instructions. The processor has an array of pipeline execution units and associated heuristics that effect how instructions are processed within the units. See for example, FIG. 1. A thread controller monitors processing of instructions within the pipeline execution units and switches between multiple program threads based upon the heuristics and urgencies of the program threads. These urgencies are based upon one or both of (a) the progress of the threads through the pipeline execution units and (b) the expected progress of the program threads through the pipeline execution units. See paragraphs [0007] and [0018].

(6) Grounds for rejection to be reviewed on appeal.

A. Whether claims 1-19 are anticipated by Keckler in accordance 35 U.S.C. §102(b).

(7) Argument.

Argument A

Keckler does not teach or suggest each and every claim limitation within claims 1-19 as required by 35 U.S.C. § 102(b).

To anticipate a claim, Keckler must teach every element of the claim and "the identical invention must be shown in as complete detail as contained in the ... claim." *MPEP 2131* citing *Verdegaal Bros. V. Union Oil Co. of California*, 814 F.2d 628, 2 USPQ2d 1051 (Fed. Cir. 1987) and *Richardson v. Suzuki Motor Co.*, 868 F2.d 1226, 9 USPQ2d 1913 (Fed. Cir. 1989). Keckler does not teach every element of claims 1-19.

As shown in Fig. 3, the MAP chip of Keckler has three clusters (processors). Table 2 of Keckler, page 909, shows that each cluster has six thread slots. That is, these six thread slots share execution units within a cluster. In col. 1, page 908, paragraph 3.2, Keckler discloses that the Multi-ALU Processor (MAP) chip implements multithreading by interleaving instructions from different threads over the execution resources of each cluster on a cycle-by-cycle bases. Keckler further discloses that "thread selection is based upon the availability of an instruction's register operands." In col. 1, page 908, paragraph 3.2.1, Keckler discloses that, in each cluster, a synchronization stage is dedicated to thread scheduling based upon a register scoreboard that indicated availability of operands for all instructions waiting in the reservation stations. We understand that to mean that the synchronization stage of Keckler schedules a thread for execution through the execution units of its cluster based upon availability of operands for the thread's next instruction. Since the instructions of Keckler are waiting within the reservation stations, the threads are already allocated to slots (i.e., pipelines) of the cluster, and, therefore, the scheduling of Keckler does not involve switching of threads in and out of an instruction pipeline. Keckler makes no disclosure of thread selection for clusters, or for slots within a cluster, based upon progress of threads through pipelines. Keckler makes no disclosure of monitoring a thread's progress. The 'operand indicators' of Keckler do not indicate progress of a thread through an instruction pipeline; the operand indicators of Keckler simply indicate the availability of operand data for a next instruction of the thread. Keckler discloses that "once ***all*** the operands are available, instructions from different threads compete for use of the execution units based on policies of priority and nonstarvation." [*Emphasis added*] See Keckler page 908, col. 1, paragraph 3.2.1. Clearly an instruction cannot progress through the execution units until **all** operand data for that instruction are available. The operand indicators

specifically indicate availability of operand data for one instruction of a thread pending in a pipeline (slot). Indication of an instruction's operand data availability, and therefore the possibility of the instructions selection for execution, is not an indication of ***a thread's*** progress through pipeline execution units, since even when all operand data is are available for an instruction, the instruction may not be selected for execution. The operand indicator of Keckler clearly only indicates operand data availability for the next instruction of a thread and does not relate to progress of a thread. The priority disclosed by Keckler is static. Keckler discloses that "this priority scheme enables critical threads, including those dedicated to event handling, to use a higher fraction of the execution resources than user-level threads." See Keckler page 908, paragraph 3.2.1. Keckler does not disclose modifying priority of a thread.

On the other hand, the '670 Application teaches of a processing unit 10, shown in FIG. 1, which may be part of an Explicitly Parallel Instruction Computing ("EPIC") processor for processing multiple program threads 15 through multiple pipeline execution units 12. See paragraph [0016]. As shown in FIG. 1, thread controller 30 includes urgency indicators 32, one per thread, that are used to determine if a thread should be switched out of its pipeline execution units to allow another thread to be switched in. These urgency indicators may be modified based upon how well a thread is progressing through a pipeline. In one example, if a thread repeatedly has cache misses, controller 30 may repeatedly lower the urgency of the thread. See paragraph [0020]. Thus the urgency indicators are based upon the progress of the ***thread***, and not based upon whether an instruction is blocked because its operand data is not available. A thread with a low urgency may thus be switched out of a pipeline to allow a thread with a higher urgency to be switched in. That is, urgency indicators determine **thread switching** within a pipeline. See paragraphs [0019], [0021] and [0026] of the '670 Application. A high urgency thread may be activated in place of a currently active low urgency thread, the low urgency thread being deactivated. If a thread stalls, the urgency of the thread may be reduced; but this does not result in the thread being switched out of the pipeline if its urgency is still higher than that of any inactive thread. The Examiner appears to use hindsight when interpreting Keckler with respect to thread urgency, since Keckler makes no disclosure of thread urgency.

Claim 1

Claim 1 recites a method for determining thread switch points within pipeline execution units of a processor, including:

a) monitoring instruction processing of a first thread within the pipeline execution units;

b) in the event of a possible switch point within the pipeline execution units, deactivating the first thread, or not, based upon a first urgency indicator for the first thread, the first urgency indicator being based upon progress of the first thread within the pipeline execution units.

Step a) of claim 1 recites that instruction processing of a first thread is monitored. Step b) recites that in the event of a possible switch point within the pipeline execution units, the first thread is deactivated, or not, based upon a first urgency indicator, which is based upon progress of the first thread within the pipeline execution units. As disclosed in paragraph [0026] of the '670 application, a thread's urgency is assessed to decide whether the current thread should be switched out of the pipeline or not. As noted above, the thread scheduling of Keckler simply selects a waiting instruction with all operand data available for execution; Keckler does not disclose thread switching within pipelines based upon urgency of the thread.

As noted above, the operand indicator of Keckler indicates availability of operand data for an instruction. Keckler does not disclose or suggest an urgency indicator based upon progress of a thread through a pipeline, and does not disclose or suggest that deactivation of the thread based upon the urgency indicator. In the '670 Application, deactivation of a thread clearly means switching the thread out of the pipeline. See paragraph [0026] of the '670 Application. Teaching away from claim 1, Keckler discloses that "unlike block multithreading [1], [34], which switches threads on long latency operations, such as cache misses, and round robin thread scheduling [2], [26], which forces a thread switch every cycle, the MAP chip makes its thread selection based upon the availability of an instruction's register operands." See Keckler page 908, column 1, paragraph 3.2 *[Emphasis added]*. Clearly, Keckler's thread selection is based upon an instruction's operand data availability, or lack thereof, and is not based upon progress of a thread. Readiness of an instruction to

proceed to an execution unit does not indicate progress of a thread. For example, in the system of Keckler, a stalled instruction may result in a thread not being scheduled. On the other hand, a stalled instruction of the '670 Application may reduce the urgency of the thread, but does not necessarily result in the thread being switched out of a pipeline execution unit. See for example paragraphs [0020], [0024-26] of the '670 Application.

In paragraph 4 of the Office Communication dated May 5th, 2006, the Examiner asserts that indication of all instruction operand data availability is based upon the progress of the thread within the pipeline execution units. Respectfully we disagree. Keckler shows no relationship between an instruction's operand availability and progress of a thread through a pipeline. Rather, Keckler discloses that "if an instruction's input data is not available, the instruction will not execute. Thus the unavailability of an instruction's operands causes the pipeline to stall, and is clearly not an indicator of a ***thread's*** progress through the pipeline.

Again, teaching away from claim 1, Keckler discloses a "simple three state priority scheme" that "enables critical threads ... to use a higher fraction of the execution resources..." See Keckler, page 908, paragraph 3.2.1. The priority system of Keckler is static and depends upon the application of the thread (i.e., a user thread, a system thread, etc). Keckler's thread priorities do not change and are not based upon progress of the thread within the pipeline execution units and therefore cannot anticipate claim 1.

As noted above, Keckler does not disclose deactivating or activating threads based upon urgency indicators. The operand data availability indicators of Keckler are not urgency indicators, and Keckler does not switch (deactivate or activate) threads within a pipeline based upon them. Keckler does not disclose urgency indicators and therefore cannot disclose or suggest 'deactivating the first thread, or not, based upon a first urgency indicator for the first thread' as required by claim 1. Reconsideration of claim 1 is requested.

Claims 2-13 depend from claim 1 and benefit from like arguments; but in addition these claims have other features that patentably distinguish over Keckler.

Claim 2

Claim 2 recites deactivating the first thread and activating a second thread based upon a second urgency indicator for the second thread, the second urgency indicator being based upon expected progress of the second thread within the pipeline execution units. As noted above, the priority system of Keckler is not equivalent to urgency indication of the '670 Application. In Keckler, priority is fixed and is not dependent upon thread progress through a pipeline. Keckler does not disclose or suggest using a second urgency indicator of a second thread to decide upon deactivating the first thread and activating the second thread, as required by claim 2. As noted above, the thread scheduling of Keckler does not switch threads in and out of pipelines; it simply selects, from a pipeline, an instruction that is ready for execution by execution units of a cluster.

Claim 3

Claim 3 recites deactivating the second thread, or not, based upon the second urgency indicator for the second thread and in the event of a possible switch point event of the second thread. Keckler does not disclose deactivating the second thread, or not, based upon the second urgency indicator for the second thread and in the event of a possible switch point event of the second thread, as required by claim 3. Keckler does not disclose thread switching (deactivating and activating of threads within a pipeline) based upon thread urgency or thread progress.

Claim 4

Claim 4 recites activating another thread within the pipeline if the second thread is switched out. Keckler does not disclose or suggest activating another thread within the pipeline if the second thread is switched out, as in claim 4. Keckler does not disclose thread switching within a pipeline based upon thread urgency.

Claim 5

Claim 5 recites deactivating the first thread, or not, based upon the first urgency indicator and upon a second urgency indicator of a second thread, the second urgency indicator being based upon expected progress of the second thread within the pipeline execution units. Keckler does not disclose an urgency indicator being based upon expected progress of a thread. Keckler does not disclose switching (deactivating and activating threads within a pipeline) threads based upon urgency of the thread.

Claim 6

Claim 6 recites utilizing a thread controller coupled with the execution units. The Examiner asserts that the synchronization stage of Keckler represents the thread controller coupled with the execution units. Respectfully we disagree. The thread controller of the '670 Application may switch threads in a pipeline. The synchronization stage of Keckler does not switch threads in a pipeline, but instead determines if an instruction's operand data is available such that the instruction may be executed. Keckler's synchronization (SZ) stage, on the other hand, is a dedicated pipeline stage. Keckler discloses "one 3-wide instruction waits in the SZ stage's reservation station until the instruction is ready to issue." See Keckler paragraph 3.2.1, page 908. Clearly, Keckler's synchronization stage operates upon instructions, and does not monitor instruction processing of a first thread.

Claim 7

Claim 7 recites modifying the first urgency indicator to increase or alternatively decrease urgency of the first thread based upon characteristics associated with the possible switch point. Keckler does not teach or suggest urgency indicators for a thread and does not teach or suggest modifying urgency indicators based upon characteristics associated with possible switch points of the thread.

Claim 8

Claim 8 recites determining whether a time slice expiration occurred. Keckler does not disclose determining whether a time slice expiration occurred. In fact, as noted above, Keckler teaches away from claim 8, stating "the MAP chip makes its thread selections based upon the availability of an instructions register operands." See Keckler page 908, paragraph 3.2. The Examiner asserts that the preempt state of Keckler is equivalent to time slicing. Respectfully, we disagree. The preempt state of Keckler guarantees "that a ready instruction can only be stalled for up to 255 cycles." See Keckler page 908, paragraph 3.2.1. As known in the art, time slicing provides a method of ensuring equal periods of processor utilization between threads. We contend that the preempt state of Keckler is not equivalent, at lease because this preempt state does not divide processor utilization equally between threads, and is only used when "one thread is creating a tremendous number of events," such that

"other threads will be able to continue making forward progress." See Keckler pages 908-909, end of paragraph 3.2.1.

Claim 9

Claim 9 recites utilizing a time slice expiration unit. As argued above, the preempt state of Keckler is not equivalent to time slicing, and therefore Keckler's preempt state cannot represent a time slice expiration unit.

Claim 10

Claim 10 recites determining whether a cache miss occurred. Again, teaching away from claim 10, Keckler states "Unlike block multithreading ... which switches threads on long latency operations, such as cache misses ... the MAP chip makes its thread selections based upon the availability of an instructions register operands." See Keckler page 908, paragraph 3.2.Thus, Keckler has no need to detect cache misses, since operation of the MAP chip does not depend upon them.

Claim 11

Claim 11 recites inserting an instruction to the pipeline to change urgency of the thread. Keckler has no teaching or suggestion of modifying a thread's urgency indicator by inserting an instruction to the pipeline, as required by claim 11.

Claim 12

Claim 12 recites deactivating the first thread and activating a second thread, and modifying urgency of the second thread. As argued above, Keckler does not modify a thread's urgency. Further, Keckler does not disclose inserting an instruction into the pipeline to modify a threads urgency.

Claim 13

Claim 13 recites monitoring possible switch points of an inactive thread having a second urgency indicator that is based upon expected progress of the inactive thread within the pipeline execution units, and deactivating the first thread, or not, based upon a first and second urgency indicators. As argued above, the priority system of Keckler is not equivalent to the urgency indicator of the immediate application. The priority system of Keckler is static and is not based upon expected progress of a thread. The priority system of Keckler provides a selection decision between threads based upon this fixed priority which is not based upon expected progress of a thread. As noted above, urgency indicators of the '670 Application are

based upon determined progress of an active thread though a pipeline and expected progress of an inactive thread if made active. In the priority system of Keckler, a high priority thread may stall frequently and therefore have less throughput than a low priority thread that does not stall. Thus, Keckler's priority is not equivalent to urgency of the '670 Application.

For at least these reasons, Keckler cannot anticipate claims 2-13. Reconsideration of claims 2-13 is respectfully requested.

Claim 14

Claim 14 recites a processor for processing multi-threaded program instructions, including:

i. an array of pipeline execution units and associated heuristics affecting how the instructions are processed within the units; and

ii. a thread controller for monitoring processing of the instructions within the units and for switching between multiple program threads based upon (a) the heuristics and (b) urgencies of the program threads;

iii. wherein the urgencies are based upon one or both of (a) progress of the threads through the pipeline execution units and (b) expected progress of the program threads through the pipeline execution units.

Keckler does not disclose or suggest a thread controller for monitoring processing of the instructions within pipeline execution units and for switching between multiple program threads based upon heuristics and urgencies of the program threads, as required by element ii of claim 14. Instead, as noted above, Keckler's multithreading is implemented by interleaving instructions from different threads over the execution resources of each cluster on a cycle-by-cycle bases. Keckler specifically teaches away from claim 14 by reciting that "unlike block multithreading [1], [34], which switches threads on long latency operations, ... the MAP chip makes its thread selections based upon the availability of an instruction's register operands." See Keckler page 908, paragraph 3.2.

By way of comparison, multithreading within the immediate application switches threads, or not, based upon one or more threads 'urgencies', which may be based upon "a thread instruction missing the cache" or "processor interrupts" (see

paragraphs [0006] and [0007]). As argued above, Keckler's thread priority is not equivalent to thread urgency of claim 14. Also as argued above, in Keckler, indication of whether all of the data for the instruction has arrive is not equivalent to an urgency indicator as required by claim 14. Again, as argued above, thread priority of Keckler does not indicate expected progress of a thread; it merely gives a priority ordering of thread selection and is not based upon expected throughput of the thread; a high priority thread can still stall in the pipeline and have lower throughput than a low priority thread.

Reconsideration of claim 14 is requested.

Claims 15-19 depend from claim 14 and benefit from like arguments; but in addition, these claims also have features that are patentably distinct from Keckler.

Claim 15

Claim 15 recites one or more of time slice expiration heuristics, cache miss heuristics and processor interrupt heuristics. Keckler does not disclose or suggest time slice expiration heuristics as in claim 15. 'Time slicing' is a mechanism by which running threads are preempted at fixed intervals to ensure that every thread is allowed time to execute. The preempt state of Keckler is not equivalent to a time slice; in Keckler, the preempt state may not occur at all. The features of claim 15, in combination with claim 14, are not anticipated by Keckler.

Claim 16

Claim 16 recites one of the instructions changing urgency for at least one thread of the processor. Keckler does not disclose or suggest program threads with one of the instructions changing urgency of at least one thread of the processor, as in claim 16. In paragraph 36 of the Office Communication dated May 5th, 2006, the Examiner argues that, in Keckler, a higher priority instruction becomes ready to execute and the urgency of the instruction increases, thereby effectively decreasing urgency of the other threads. However, this is the Examiner's interpretation of Keckler, since in fact Keckler makes no anticipating disclosure. More specifically, as noted above, the scheduling method of Keckler is instruction based and not thread based.

Claim 17

Claim 17 recites the controller modifying an urgency of any of the threads to modify future treatment of the threads in switch out events. Keckler does not disclose or describe a controller modifying an urgency of any of the threads to modify future treatment of the threads in switch out events. And Keckler does not disclose that priority of a thread is modified, nor urgency of a thread. In paragraph 21 of the Office Communication dated May 5th, 2006, the Examiner asserts that after 255 cycles, the urgency of the Keckler's instruction is modified in order to allow the thread to execute. But, Keckler does not disclose or "use urgency" or of modifying "thread urgency"; Keckler simply states "the synchronization stage also implements a preempt state (that includes idle cycle counters and limit registers) to guarantee that a ready instruction can only be stalled for up to 255 cycles, depending on the parameter stored in the limit register." See Keckler page 908, first column, paragraph 3.2.1. The Examiner appears to interpret Keckler with hindsight with regard to modifying a thread's urgency – Keckler makes no disclosure of doing anything when guaranteeing that a ready instruction can only be stalled for up to 255 cycles. Further, as argued above, the priority of Keckler is not equivalent to urgency as used in claim 17. The priority disclosed by Keckler is static; Keckler does not disclose modifying priority of a thread, and therefore Keckler's priority cannot be equivalent to urgency of the '670 Application.

Claim 18

Claim 18 recites the controller either decreasing or increasing urgency for the program threads by injecting an instruction to the pipeline execution units. Keckler does not disclose a controller decreasing or increasing the urgency (or priority) of program threads by injecting an instruction into the pipeline execution units, as required by claim 18. In paragraph 38 of the Office Communication dated May 5th, 2006, the Examiner asserts that "in order for the urgencies (priorities) to change in value, they must be instructed to do so. Changing the urgency of the thread is necessarily performed by an instruction." However, Keckler makes no such disclosure, and the Examiners assertion therefore appears to be conjecture in impermissible hindsight.

Claim 19

Claim 19 recites a time slice expiration unit for monitoring expiration of threads within the processor. Keckler does not disclose or suggest a time slice expiration unit for monitoring expiration of threads within the processor, as required by claim 19; in fact, Keckler does not operate with time slicing. In paragraph 39 of the Office Communication dated May 5th, 2006, the Examiner appears to assert that the preempt state, idle counters and limit registers of Keckler represent a time slice expiration unit. However, we interpret the preempt state of Keckler to be a mechanism for identifying a ready instruction that is stalled, and not a time slice expiration unit. Keckler does not disclose time slice operation, and as noted above, does disclose that "the MAP chip makes thread selection based upon the availability of an instruction's register operands." See Keckler page 908, paragraph 3.2.

At least for the above reasons, Keckler cannot anticipate claims 15-19. Reconsideration of claims 15-19 is respectfully requested.

In view of the above arguments, claims 1-19 patentably distinguish over Keckler. Generally, Keckler describes only how to select which thread become active at an event and does not disclose monitoring a thread's progress to determine thread switching. In accord with the inventions of claims 1-19, on the other hand, active and inactive events may be examined and may or may not result in a thread switch based on determination of one or more thread's urgency.

Claims 1-19 are therefore deemed allowable.

(8) Claims appendix.

Appellants enclose a copy of the claims involved in this appeal as an appendix hereto.

(9) Evidence appendix.

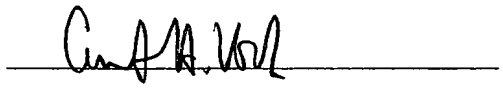
Not applicable.

(10) Related proceedings appendix.

Not applicable.

Other than the costs for this appeal brief, no further fees are deemed due in connection with this matter. However, the Commissioner is hereby authorized to charge any fees which may be due in this matter from Deposit Account Number 08-2025.

Respectfully submitted,
LATHROP & GAGE LC

By: 

Curtis A. Vock, Reg. No. 38,356
4845 Pearl East Circle, Suite 300
Boulder, Colorado 80301
Telephone: (720) 931-3011
Facsimile: (720) 931-3001

APPENDIX TO APPEAL BRIEF

1. (Previously Presented) A method for determining thread switch points within pipeline execution units of a processor, comprising the steps of:

monitoring instruction processing of a first thread within the pipeline execution units;

in the event of a possible switch point within the pipeline execution units, deactivating the first thread, or not, based upon a first urgency indicator for the first thread, the first urgency indicator being based upon progress of the first thread within the pipeline execution units.

2. (Previously Presented) A method of claim 1, further comprising deactivating the first thread and activating a second thread based upon a second urgency indicator for the second thread, the second urgency indicator being based upon expected progress of the second thread within the pipeline execution units.

3. (Original) A method of claim 2, further comprising deactivating the second thread, or not, based upon the second urgency indicator for the second thread and in the event of a possible switch point event of the second thread.

4. (Original) A method of claim 3, further comprising activating another thread within the pipeline if the second thread is switched out.

5. (Previously Presented) A method of claim 1, the step of deactivating the first thread comprising deactivating the first thread, or not, based upon the first urgency indicator and upon a second urgency indicator of a second thread, the second urgency indicator being based upon expected progress of the second thread within the pipeline execution units.

6. (Original) A method of claim 1, the step of monitoring comprising utilizing a thread controller coupled with the execution units.

7. (Previously Presented) A method of claim 1, further comprising modifying the first urgency indicator to increase or alternatively decrease urgency of the first thread based upon characteristics associated with the possible switch point.

8. (Original) A method of claim 7, further comprising determining whether a time slice expiration occurred.

9. (Original) A method of claim 8, further comprising utilizing a time slice expiration unit.

10. (Original) A method of claim 7, further comprising determining whether a cache miss occurred.

11. (Original) A method of claim 7, further comprising inserting an instruction to the pipeline to change urgency of the thread.

12. (Original) A method of claim 1, further comprising the steps of deactivating the first thread and activating a second thread, and modifying urgency of the second thread.

13. (Previously Presented) A method of claim 1, further comprising the steps of monitoring possible switch points of an inactive thread having a second urgency indicator that is based upon expected progress of the inactive thread within the pipeline execution units, and deactivating the first thread, or not, based upon a first and second urgency indicators.

14. (Previously Presented) A processor for processing multi-threaded program instructions, comprising:

an array of pipeline execution units and associated heuristics affecting how the instructions are processed within the units; and

a thread controller for monitoring processing of the instructions within the units and for switching between multiple program threads based upon (a) the heuristics and (b) urgencies of the program threads;

wherein the urgencies are based upon one or both of (a) progress of the threads through the pipeline execution units and (b) expected progress of the program threads through the pipeline execution units.

15. (Previously Presented) A processor of claim 14, the heuristics comprising one or more of time slice expiration heuristics, cache miss heuristics and processor interrupt heuristics.

16. (Previously Presented) A processor of claim 14, the program threads comprising one or more instructions, one of the instructions changing urgency for at least one thread of the processor.

17. (Previously Presented) A processor of claim 14, the controller modifying an urgency of any of the threads to modify future treatment of the threads in switch out events.

18. (Previously Presented) A processor of claim 17, the controller either decreasing or increasing urgency for the program threads by injecting an instruction to the pipeline execution units.

19. (Previously Presented) A processor of claim 14, further comprising a time slice expiration unit for monitoring expiration of threads within the processor.

EVIDENCE APPENDIX

(Not applicable)

RELATED PROCEEDINGS APPENDIX

(Not applicable)